# THE ISIS PROJECT: Final Technical Report

**Project term: February 5, 1985 - May 31, 1987**

**Kenneth P. Birman**



This work was sponsored by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 5378, under contract MDA903-85-C-0124 issued by the department of the Army.

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION / AVAILABILITY OF REPORT<br>Approved for Public Release<br>Distribution Unlimited |
| 2b DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION<br>Kenneth P. Birman, Assist. Prof<br>CS Dept., Cornell University<br>6b OFFICE SYMBOL (If applicable) | 7a NAME OF MONITORING ORGANIZATION<br>Defense Advanced Research Projects Agency/ISTO |
| 6c. ADDRESS (City, State, and ZIP Code) | 7b. ADDRESS (City, State, and ZIP Code)<br>Defense Advanced Research, Project Agency<br>Attn: TIO/Admin, 1400 Wilson Blvd.<br>Arlington, VA 22209-2308 |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION<br>DARPA/ISTO<br>8b OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State, and ZIP Code)<br>See 7b. | 10 SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO |

11 TITLE (Include Security Classification)
The ISIS Project: Final Technical Report

12 PERSONAL AUTHOR(S)
Kenneth P. Birman

| 13a TYPE OF REPORT | 13b TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| Technical (Special) | FROM 2/5/85 TO 6/31/87 | September 1, 1987 | 12 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | |
| | | | |
| | | | |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report summarizes the accomplishments of the ISIS project during the entire funding period. It assumes some knowledge regarding our overall effort.

| | |
|---|---|
| 20 DISTRIBUTION / AVAILABILITY OF ABSTRACT<br>☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION |
| 22a NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

## Academic Staff

**Kenneth P. Birman, Principal Investigator**

**Thomas A. Joseph, Research Associate**

## Graduate Students

**Amr. El. Abbadi**

**Richard Koo**

**Kenneth Kane**

**Frank Schmuck**

**Ajei Gopal**

## 1. Description of Progress

This report summarizes the accomplishments of the ISIS project during the entire funding period. It assumes some knowledge regarding our overall effort.

## 2. Summary of activities on technical areas identified in the initial contract.

During the first two and one-half years of DARPA funding of the ISIS project our effort was focused on support for *resilient objects*: abstract data types capable of tolerating failures. Below, we list the major tasks that we were asked to address in connection with this concept. In each case, the actual accomplishments of the project are discussed.

1.) *Techniques for efficiently implementing resilient objects.* It is increasingly clear that existing methods for producing fault-tolerant software are inadequate. Although several groups claim to have developed such technologies, the fact that so few distributed systems actually exist argues that none has yet achieved wide success. Distributed applications that do exist are generally intolerant of failures, and are often surprisingly non-distributed in their internal architecture. For example, the UNIX network file system (NFS) maintains no distributed state information, and the ARPANET mail and bulletin board facilities are notoriously unreliable. Yet, few other distributed "application" programs of any kind exist.

Several projects have attempted to address this shortcoming. At MIT, the ARGUS project developed a transactional language for fault-tolerant distributed computing. However, this approach does not (yet) provide any help if the goal is to replicate data. The V project, at Stanford, developed a system based on inexpensive remote procedure calls and the grouping of programs providing a service into *process groups* that clients can access with no knowledge of the current group membership. However, V provides little help in tolerating failures. ISIS-1, the first version of the ISIS system, adopted an approach that combines elements of these two systems: a resilient object is programmed using a

language similar to ARGUS, but compiles into a distributed program capable of functioning correctly despite site and program crashes and concurrent requests from application programs. Our approach was uniquely transparent: programmers who construct a resilient object describe it as if it were not distributed and failures never happen: replication of data, failure handling, and recovery mechanisms are all provided by the ISIS-1 system. Similarly, clients issue RPC's to a resilient object as if it were a single instance of the specified abstract datatype executing at some single site; actual execution of the request is automatically handled by ISIS-1.

2. *Procedures for improving resilient object performance.* Performance is one of the basic problems we confronted in connection with our resilient object approach, and we developed a unique *concurrent updating* mechanism for dealing with this issue. The mechanism is based on a new asynchronous broadcast primitive, the *causal broadcast*, which has turned out to be one of the most important contributions of our two year effort. By combining this mechanism with several others, we achieved replicated update performance that was roughly as fast as a single-site read request would be in a typical system. This performance exceeds that which can be obtained with any other approaches. On the other hand, like ARGUS, resilient objects suffer from some insurmountable forms of overhead. What we found is that because resilient objects use a transactional correctness constraint, performance of such an object can never be as good as for a system that can get away with some cheaper correctness constraint. Moreover, because resilient objects are discrete entitied, and are not compiled directly into their clients, the costs of just getting a request to them can be high -- measured in milliseconds in many cases. We return to this issue below. To summarize, because of our concurrent update algorithm, resilient object performance is extremely good -- far better than we imagined could be possible. However, performance could be still better if non-transactional correctness constraints could be used instead of the ARGUS-like transactional one we adopted, and if the mechanisms could be unbundled from the rigid "abstract type" framework in which it

was packaged.

3. *Techniques for minimizing inter-site message traffic in a system supporting resilient objects.* In this area, we developed a number of techniques for asynchronous broadcasting and piggybacking. These are interesting because they can boost performance considerably above the level that can be achieved using a "greedy" point to point RPC interaction. The approach has lead us to develop a suite of broadcast primitives that are integrated into a package, and to code algorithms that exploit the cheapest possible primitive for each type of activity they undertake. The causal broadcast mentioned above is one of these; the suite covers a full range of broadcast types. This approach runs contrary to the accepted wisdom in distributed systems design, which has been increasingly oriented towards support for efficient RPC mechanisms. As a result, we are deeply involved in efforts to develop new message transport protocols capable of efficiently supporting broadcast interactions between sites in a distributed system.

4. *A prototype version of the ISIS-1 system capable of supporting simple objects.* A prototype has existed for more than two years, and has been demonstrated to our DARPA program director (currently, Dr. Dennis Perry). A second generation system, quite different from this first system, but preserving a form of upward compatibility, is now under development. We describe it further below.

5. *Evaluation of the prototype.* A performance evaluation of the ISIS-1 prototype was undertaken, and reported in [2]. This evaluation was based on some simple application software for maintaining a distributed database of appointments (a calendar) and for a simple process control application. The applications performed well enough to confirm that our concurrent update mechanism yields excellent replicated data update performance. At the same time, the absolute performance of the system was not as high as we would like.

Systematic study has revealed that the overhead in question stems primarily from two aspects. First, the system is transactional and the associated concurrency control mechanisms are costly. Secondly, resilient objects exist independent of their clients. A consequence is that one cannot access such an object without sending it a message, and even when the client and the object are coresident at a single site, this imposes substantial costs. Moving from UNIX to a system like V might reduce these costs. Overall, however, a more basic system restructuring seems to be indicated.

## 3. The distributed systems tool kit

Early this year we began work on a new system that we expect to complete this summer, and which is replacing the ISIS-1 prototype in our experimental work here at Cornell. We call the new system ISIS-2. ISIS-2 retains the mechanisms that worked best in ISIS-1, while stripping from it the aspects that proved to be bottlenecks. The overall form of this system is as follows. At the lowest level, it consists of an implementation of a suite of broadcast communication protocols that support *virtually synchronous process groups*. This concept, which we discuss in several papers, represents a breakthrough in our approach to distributed systems construction. Like V, it is an approach oriented towards support for process groups, but unlike V, issues relating to tolerance of failures and supporting high levels of concurrency are addressed are part of the process group mechanism. For example, members of a virtually synchronous process group can migrate from site to site, or can exchange responsibility for processing some task, without any risk of a client interacting with the group before the change has completed. Because all group members receive a given message if any does, and all receive it in the same "state", the approach creates a remarkably simple environment within which to develop distributed algorithms. Moreover, performance can be extremely high -- in comparison with ISIS-1, this is a real "RISC" approach to distributed computing. At the same time, it is an environment fully capable of supporting the mechanisms used in the ISIS-1 prototype.

Few programmers have the sophistication to program at the level of asynchronous distributed broadcasts, even in an environment providing virtual synchrony (the appearance that one event happens at a time). Accordingly, we are packaging our primitives into a set of tools that will form a library of mechanisms covering all the types of actions that distributed programs need most often. A paper that we will present at the upcoming SOSP conference in Austin covers this approach, which has generated widespread interest and enthusiasm among researchers in the field who have learned of it. An implementation of the primitives was completed this month, and the first version of the toolkit is just starting to limp along. By the end of this summer, we expect to have a fully operational toolkit with some non-trivial application software running on it.

### 4. Asynchronous bulletin boards

Distributed artificial intelligence programs are often constructed using a bulletin board *paradigm whereby information is shared* through one or more common bulletin boards on which processes can post and read information at will. A bulletin board is a simple and highly asynchronous form of shared memory, and it occurred to us that these might provide a cheaper mechanism than resilient objects accomplishing much the same goal. A paper describing our approach has been written, and an implementation of the mechanism is underway as one of the "tools" in the distributed systems toolkit reported above.

### 5. The importance of compatibility

We used to believe that all distributed applications could somehow be twisted to fit into the resilient object approach -- much as the ARGUS project twists all applications into nested actions, V into process groups interacting via RPC, and LOCUS into replicated files. UNIX, which simply provides communication paths, is outstanding at providing high bandwidth data streams and even network access to file systems, but much weaker at providing fancier mechanisms on top of these streams. We no longer believe that any one approach -- even our own -- can address the needs of every possible class of system. A more realistic approach is to

concede that each of these techniques is nearly ideal for some class of applications, but that none is ideal for all classes. Thus, the ISIS-2 system may well provide a uniquely high quality of support for maintaining consistent replicated data structures, migrating tasks from site to site, and recovering from failures. Yet, ISIS-2 will inevitably have to coexist with high bandwidth stream mechanisms, transactional database mechanisms, and network file systems. This argues that the toolkit routines must be designed in such a manner as to be compatible with one another -- it should not ever be the case that the use of one routine renders another incorrect, or that the use of some other mechanism (like a high bandwidth communication channel) invalidates an assumption made by some other part of the system. ISIS-2 is being designed with this in mind. Over a period of time we will provide interfaces to a wide range of UNIX mechanisms, transactional mechanisms, etc. in such a manner as to guarantee that the correctness of all of these mechanisms is preserved regardless of how they are combined.

## 6. A serious application

At the present stage of our research, serious applications are needed to push ISIS-2 to its limits. We are collaborating with Keith Marzullo on the development of several kinds of such applications. These will be in the area of distributed file systems (we are constructing an "RFS" program that will provide file replication and fault-tolerance using normal UNIX NFS systems as its components), distributed process control (we are investigating the adaptation of some of Keith's work on clock synchronization to realtime issues within ISIS-2) and programming in the large. In light of this, we expect ISIS-2 to be supporting a moderate user community on a regular basis by the end of 1987.

## 7. Network partitioning

A final area in which we have made recent progress concerns network partitioning. The current approach to ISIS-2 is intolerant of partitioning -- partitioning failures can cause it to "hang" until the partition resolves itself. In a local area network, this will not be much of a problem because such networks simply do not partition very often. However in larger

networks, clusters of ISIS-2 sites may have to be interconnected over links that do partition, and blocking in this case is unsatisfactory. Most existing work on partitioning is oriented towards databases, which assume a transactional correctness constraint. As noted above, ISIS-2 no longer assumes this about application software. Thus, new techniques for dealing with partitioning are needed.

Working with Ajei Gopal, a new graduate student member of the project, some hope for non-blocking communication across partitionable links has now emerged. Our approach allows for a special class of long running protocols, which can be derived in a mechanical manner from conventional protocols such as the broadcast protocols used within ISIS-2. Rather than run the usual ISIS-2 protocols directly across links that can partition, these special protocols would be used in a hierarchical fashion. Good performance, rapid termination, and tolerance of partitioning results. We expect to complete a paper on this new work during the summer or fall of this year. Moreover, an implementation will be undertaken as part of our new system.

## 8. Budget

A budgetary summary for the entire period of support appears below. All expenditures are in line with projections under our original budget.

Birman - Darpa final report -
E66-8325
Darpa - MDA903-85-C-0124 DSS - 9/30/87

| Salaries and Wages | Year I Budget | Year II Budget | Total Funding | Actual Expenses |
|---|---|---|---|---|
| Principal Investigator | | | | |
| Summer | $16,600 | $17,800 | $34,400 | $34,044 |
| Academic year | 5,375 | 5,840 | 11,215 | $20,150 |
| Professionals | | | | |
| Research Associate | 12,500 | 33,000 | 45,500 | $49,352 |
| Programmer | 0 | | | $9,000 |
| Secretary | 4,200 | 4,600 | 8,800 | $10,106 |
| Graduate Students | | | | |
| Academic year | 54,810 | 50,860 | 105,670 | $129,731 |
| Summer | 16,800 | 18,000 | 34,800 | |
| Ugrad | 4,115 | 4,420 | 8,535 | |
| Total Salaries and Wages | 114,400 | 134,520 | 248,920 | 252,383 |
| Employee Benefits | | | | |
| Summer | 1,660 | 1,780 | 3,440 | $3,404 |
| Academic year | 6,158 | 12,272 | 18,430 | 24,527 |
| General Expenses | | | | |
| Travel domestic | 6,000 | 7,000 | 13,000 | $14,044 |
| Miscellaneous | | | | |
| Supplies | 3,600 | 4,000 | 7,600 | $7,792 |
| Publications | 2,400 | 2,400 | 4,800 | $2,857 |
| Computer supplies | 2,000 | 2,000 | 4,000 | $643 |
| Computer maintenance | 7,200 | 7,500 | 14,700 | $4,035 |
| Lecturer fees | | | | $834 |
| Equipment | 68,000 | | 68,000 | $76,360 |
| Indirect cost | 72,876 | 94,310 | 167,186 | 163,197 |
| Total | $284,294 | $265,782 | $550,076 | 550,076 |

NOTE: Any deviation from the original budget received prior approval from Dennis Perry.

## 9. Bibliography

Below, we list all publications of the project during the period of DARPA funding,


[1] K. Birman, T. Joseph, T. Raeuchle, and A. El Abbadi. Implementing Fault-Tolerant Distributed Objects. IEEE Transactions on Software Engineering, TSE-11, 6, (June 1985), 502-508,

[2] K. Birman. Replication and Fault-Tolerance in ISIS. 10th ACM Symposium on Operating Systems Principles, December 1985.

[3] T. Joseph, K. Birman. Low-Cost Management of Replicated Data in Fault-Tolerant Distributed Computing Systems. ACM Trans. on Computing Systems, 4,1 (Feb. 1986), 54-70.

[4] T. Joseph. Birman. Low-Cost Management of Replicated Data in Fault-Tolerant Distributed *PhD. dissertation*, Cornell University, December 1985.

[5] K. Birman, T. Joseph. Exploiting virtual synchrony in distributed systems. Department of Comp. Sci., Cornell University, TR 87-811 (Feb. 1987), *submitted to: ACM SIGOPS SOSP.*

[6] K. Birman. Communication Support for Reliable Distributed Computing. Proc. Asilomar Workshop on Fault Tolerant Distributed Computing, March 1986.

[7] T. Raeuchle. Concurrency control for libraries of types objects. *PhD. dissertation*, Cornell University, June 1986.

[8] A. El Abbadi, S. Toueg. A paradigm for distributed replicated database protocols. Department of Comp. Sci., Cornell University, *forthcoming. submitted to: ACM SIGACT/SIGOPS PODC.*

[9] A. El Abbadi A paradigm for distributed replicated database protocols. *PhD. dissertation*, Cornell University, August 1987.

[10] K. Birman, T. Joseph, P. Stephenson, F. Schmuck. Programming with asynchronous bulletin boards in distributed systems. Department of Comp. Sci., Cornell University, TR

86-772, (August 1986; revised December 1986), *submitted to: ACM TOCS.*

[11] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM TOCS* **6**, 1 (Feb. 1987).

[12] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. To appear, Proc. 11th ACM Symposium on Operating Systems Principles, November 1987.